(12)

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 86-07-02	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) MOS Circuit Models in Network C		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) William Beckett		8. CONTRACT OR GRANT NUMBER(s) MDA903-85-K-0072 ARPA-4563, #2 Code 5D30
9. PERFORMING ORGANIZATION NAME AND ADDRESS UW/NW VLSI Consortium, Dept. of Computer Sci. University of Washington, FR-35 Seattle, WA 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA - IPTO 1400 Wilson Boulevart Arlington, VA 22209		12. REPORT DATE July 1986
		13. NUMBER OF PAGES 8
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ONR University of Washington 315 University District Building 1107 NE 45th St., JD-16, Seattle, WA 98195		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DTIC ELECTE JUL 24 1986 S D

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Network C, VLSI, MOS, CMOS, nMOS, SPICE, nmosfet, pmosfet, simulation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Network C is a programming language designed for constructing simulation models of VLSI circuits and systems. The language, which is a superset of C, supports a range of modeling capabilities including approximate solution of Kirchoff equations at the circuit level and discrete event functional simulation at the system level. When used to model a MOS circuit, the system first decomposes the circuit into a set of independent stages. The values of nodes, represented by piece-wise linear functions, are communicated between stages using discrete event scheduling. The determination of these piece-wise linear (continued over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

20. Abstract (continued)

functions is based on continuous time calculations. The result of this hybrid approach is a fast simulation capability which maintains enough accuracy to capture both the digital and analog aspects of a circuit's behavior.

This paper bears on the topic of simulation.

MOS Circuit Models in Network C

William Beckett

University of Washington
Seattle, WA 98195

Technical Report 86-07-02
July 1986

MOS Circuit Models in Network C

Abstract

Network C is a programming language designed for constructing simulation models of VLSI circuits and systems. The language, which is a superset of C, supports a range of modeling capabilities including approximate solution of Kirchoff equations at the circuit level and discrete event functional simulation at the system level. When used to model a MOS circuit, the system first decomposes the circuit into a set of independent stages. The values of nodes, represented by piece-wise linear functions, are communicated between stages using discrete event scheduling. The determination of these piece-wise linear functions is based on continuous time calculations. The result of this hybrid approach is a fast simulation capability which maintains enough accuracy to capture both the digital and analog aspects of a circuit's behavior.

This paper bears on topic 1 (Simulation).

MOS Circuit Models in Network C

Introduction

Network C is a programming language designed for constructing simulation models of VLSI circuits and systems. The language, which is a superset of C, supports a range of modeling capabilities including approximate solution of Kirchoff equations at the circuit level and discrete event functional simulation at the system level. The circuit description capabilities of the language are hierarchical and allow subsystem models of varying levels of precision to be mixed.

In the case of MOS models, the execution of a Network C program has two phases. The first phase is circuit analysis. The effect of circuit analysis is the decomposition of the system being modeled into a set of stages. Each stage is isolated in that the only connections existing between it and all other stages are through nodes connected to gates. After the stages have been isolated in this fashion, the approximate behavior of the system can be obtained by evaluating each stage independently.

The calculation phase of Network C utilizes a combination of continuous time calculation and discrete event scheduling. This technique is aimed at retaining some of the accuracy of purely continuous time systems while realizing the speed advantage inherent in discrete event systems. The value of increased accuracy over purely discrete systems is that a larger class of circuits can be modeled. For example, circuits which utilize analog circuit techniques or in which there is a considerable amount of charge sharing are usually beyond the capabilities of purely switch level simulators. The value of the increased speed of discrete event systems over purely continuous time systems is that circuits with a larger number of components can be accommdated.

Discrete event simulation requires that each stage accept state valued functions as input and produce state valued functions as output. To meet this criterion, Network C models node voltages with piece-wise linear functions. Since piece-wise linear functions can represent an unbounded number of states, Network C reduces the state space by truncating both the derivatives and values of the functions to fixed precission. This produces a set of states which, although it is small relative to a continuous representation, it is still large enough for substantially improved precission when compared to systems with only a few to tens of states.

The continuous time part of the calculation used by Network C differs from the calculation done in SPICE[4] in three fundamental respects. First, since Network C partitions the circuit into stages and computes each stage independently, the rank of the set of node equations is dramatically reduced for larger systems. Second, Network C uses direct three step quadrature rather than a nonlinear equation solver to solve the equations for each stage. Finally, Network C uses only simple DC MOS law models for transistors. All of these aspects tend to trade accuracy for speed.

While Network C may be used to model systems at high levels of abstraction, the purpose of this paper is to provide an overview of the Network C facilities used in developing circuit level models with emphasis on describing the calculation used to evaluate MOS circuits. The first section describes MOS circuit analysis and lists the assumptions about the nature of MOS circuits on which the method is based. The next section discusses behavior calculation in detail. Finally, two short examples are presented. Although a complete definition of the the syntax and semantics of the various Network C constructs is beyond the scope of this paper, many of these constructs are illustrated in the examples.

Circuit Analysis

The MOS abstraction implemented by Network C embodies the following three hypotheses.

1. MOS circuits are composed primarily of gates connected by passive steering networks. The function of the gates is to connect various circuit capacitors to the power and ground rails for charging and discharging. The interpretation of the behavior of the circuit is in terms of the voltages on these capacitors at any point in time. That is, systems are designed so that information is not directly represented by current flows.
2. The power and ground rails have zero impedance and can supply arbitrary currents.
3. The average current into the gate terminals of MOS transistors is zero. Hence, there is no possibility of DC coupling between stages of the circuit.

Assumption 2 is implemented by simply holding the voltages of all device terminals connected to the supply rails constant at the corresponding level.

The Network C uses assumption 3 as the basis for its decomposition of circuits into stages. Each stage consists of a subset of the nodes of the original circuit which may be reached from each other without crossing any gates, that is, by following only source-drain paths. These nodes are called the output nodes of a stage. All devices with either their source or drain connected to an output node in a stage are also considered part of that stage.

Circuit analysis partitions the circuit so that the value of each node in the circuit is determined by exactly one of the stages. That is, all drivers of a node, if there are more than one, belong to the same stage.

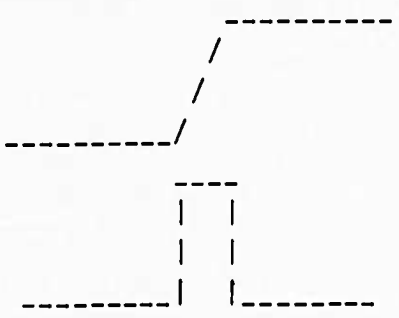
When circuit analysis has completed, each stage will contain zero or more devices whose gates are not connected to nodes determined by that stage. The voltages on these gates are considered to be the independent variables or input nodes from whose values the behavior of the output nodes of stage is computed.

Behavior Calculation

Discrete event scheduling is used to control the operation of the behavior calculation.

The models for each stage, having been derived by circuit analysis, are invoked whenever an input to the stage changes state. The effect of evaluating the model is the calculation of new descriptions for the voltages of each of the output nodes in the stage.

The inputs of a stage are the nodes that are connected to the stage's independent gates. Note that by 'change in state' is meant a change in the parameters of the linear model for a node's voltage, not simply a change in the voltage. To clarify the nature of a change in state. consider the following waveform and its derivative:



This wave is a typical rising edge. It's state changes twice. During the first change, the derivative (which is the slope of the linear model) goes from zero to a positive value, during the next change it goes back to zero. Note that the use of the edges of the derivative as events in this simulator is analogous to use of the edges of the logic level as events in logic level simulators except that, in the case of derivative edges, there are typically two events per logic state change.

The evaluation of each stage involves the determination of a piece-wise linear model for each of its output nodes given piece-wise linear models for each of its input nodes. Although more direct methods are possible for simple stages, currently Network C generates piece-wise linear models by performing continuous time calculations and then fitting the resulting curve with a piece-wise linear form.

The continuous calculations are performed forward in time from the current time point. Since the behaviour of the inputs for future time has not yet been calculated, the calculation of the output forecast is based on the *predicted* behaviour of the inputs.

This prediction of input behaviour is computed as follows. The state of each node in the circuit consists of the three parameters of its linear model, namely:

m	slope
b	intercept
t_0	time of last change

The availability of these parameters means that the value of the node at any future point in time can, in principle, be predicted using the linear formula

$$y = m(t - t_0) + b$$

Actually, the above formula works for large values of $t - t_0$ only if the signal is constant, that is $m = 0$. Otherwise, the formula generates an unbounded value. Therefore, immediately following applications of this formula, Network C bounds the result above and below by the power supply rails. The effect of this heuristic, which works better for CMOS circuits than NMOS circuits, is that the linear models for all nodes in the system are given a piece-wise linear interpretation.

The calculation of outputs from inputs proceeds as follows. For each time point in the forecast range, the system calculates branch currents for each DC branch in the stage. Note that there are no DC branch currents between stages since there are no DC paths between stages. These currents are DC branch currents because, during this part of the computation, all node capacitances are considered to be zero.

The branch currents through each transistor are calculated using the following continuous form the the DC MOS law:

$$\begin{array}{ll} i_{lin} = 2k(v_{gs} - v_{th})v_{ds} - kv_{ds}^2 & v_{gs} - v_{th} \geq v_{ds} \\ i_{sat} = k(v_{gs} - v_{th})^2 & v_{gs} - v_{th} < v_{ds} \\ i_{off} = 0 & v_{gs} - v_{th} < 0 \end{array}$$

/noindent The above equations are for NMOS; for PMOS the equations are similar.

Inputs to the calculation are the voltages on the source and drain nodes at time $t-1$, the forecast time, and the linear model of the gate node. Using the forecast time and the linear model, the gate voltage is determined by the prediction algorithm described above. Outputs from this calculation are the source and drain terminal currents which are the same except for the sign.

After all branch currents for the stage have been computed, the currents for each node are summed. This is similar to the normal Kirchoff procedure except that, since the node capacitances have been disconnected, the result of this summing is a non-zero residual current at the each node.

Next, these residual currents are smoothed using the filter:

$$i = (i_0 + 2i_{t-1} + i_{t-2})/4$$

where i_0 is the unfiltered residual current, i_{t-1} is the filtered residual current at $t-1$ and i_{t-2} is the filtered residual current at $t-2$.

The final filtered residual currents are then forced into the node capacitances producing the set of node voltages for this time point. This particular numerical technique is similar to that used in QRS[6].

The above procedure continues until a complete set of output curves for the stage has been computed. Typically this set of curves will span about 50 nano-seconds and contain 50 time points per output node.

Each of these curves is then fit using a piece-wise linear curve fitter. The curve fitter classifies curves by the number of inflections in their second differences and will try to generate a fit having one, two or three line segments. Each segment is the best linear least squares approximation for the points in the segment.

After the curve fitting, a continuity constraint is applied to the resulting piece-wise linear form: in the case in which the new piece-wise linear model for a node intersects the current piece-wise linear model at some future time, the first event in the new piece-wise linear form is delayed until that time.

Finally, the events of the new piece-wise linear model of the node are queued. There can be from one to three of these events and, since the queuing mechanism is preemptive, some events in the new form may preempt events queued earlier.

Since the determination of piece-wise linear models involves a continuous time computation, and since this computation is really a forecast which may have to be recalculated if the assumptions on which it is based change, Network C simulations could potentially require more computation than fixed time step continuous time approaches.

There are several factors that tend to overcome this tendency for increased computation. First, since the scheme is event driven, detailed calculations for any stage are performed only in the neighborhoods of transitions on nodes which are inputs of the stage. Second, when an event occurs on an input that was predicted during a previous invocation of the model of a stage, the model exits immediately and the output is not recalculated. Since the new event was anticipated, its effects have already been included in the output forecast.

Finally, redundant calculations are avoided by using a calculation history. Associated with each stage, these calculation histories consist of a small set of situations and actions. A situation is an encoding of the states of the inputs and outputs of the stage at a time immediately following a change in an input. The action is the set of outputs that were computed in response to the situation. Since the most stages tend not to have very many nodes, and since the parameters of the piece-wise linear models are granular, it is feasible to keep a reasonable number of situations in the history. Also, since the shapes of the transitions for most nodes in digital circuits are determined by time invariant physical characteristics, it is likely that the situations recur. When this happens, the output for the stage is available immediately as a result of table look up in the calculation history.

The net effect of all these mechanisms is that the resolution of the simulator is variable. In the worst case of continuously changing input, say a sinusoid, the forecast and fit procedures will generate a large number of short lived piece-wise linear models of localities of the sine wave. Of course, using this mechanism to track a sine wave is expensive. Generally speaking, the more the natural behaviour of the nodes of the circuit fit the assumptions of the forecasts the less the amount of calculation required. In the best case it is possible for the simulator to completely learn the circuit and, once that happens, all behaviour is obtained by table lookup.

Examples

In discussing the examples, a number of the features of Network C language and translator will be highlighted. The Network C translator is implemented using a three pass preprocessor. The first pass uses a *yacc* parser to build a complete parse tree. The grammar used by the parser consists of the grammar for the portable version of the C compiler with the extra Network C constructs added. The second pass applies a number of parse tree rewriting rules to convert the subtrees containing Network C constructs into subtrees containing only C constructs. The third pass walks the modified parse tree outputing the text of the C translation of the original Network C input.

The C programs generated by the Network C translator can be compiled by the system C compiler and the resulting object files represent executable models of VLSI subsystems. These models, like all C procedures, may be put on libraries for inclusion in other, more complex models.

Network C programs have two module types - network descriptions (called circuits) and procedural device or subsystem models (called models). Circuits, or network descriptions, consist of a set of elements connected to a set of nodes. An element is either another circuit or a model.

Models are procedures which compute the values of outputs from inputs.

The first example, shown in Figure 1, is a simple series of three CMOS inverters. The example illustrates the general form of Network C programs.

```
maincircuit mos9()
/*  Simplified cmos inverter chain                */
{
      elements
      {
         g1   (gen, 0.0,5.0,1.0e9,-1.0e9)  clk, a, x;
         i1   (cinv,5.0,0.0,1.0e-4,1.0e-4) a,   b;
         i2   (cinv,5.0,0.0,1.0e-4,1.0e-4) b,   c;
         i3   (cinv,5.0,0.0,1.0e-4,1.0e-4) c,   d;
      };
      conditions
      {
          clk = (clk+1) % 2; [50.e-9]
      };
}
```

Figure 1.

The declarator *elements* introduces the list of network elements. Each network element

has an instance name, a class name, a set of optional parameters, and a terminal connection list.

The first element in this circuit, *g1*, is a clock generator which generates a clock signal *a*. *a* is synchronized to the signal *clk* and has a minimum value of 0.0, a maximum value of 5.0, and rise and fall rates of 1 volt per nanosecond. The second phase of the clock, *x*, is not used in this experiment.

The next three elements form the series of inverters. The clock, *a*, drives the first inverter. Its output, *b*, drives the next inverter whose output, *c*, drives the last inverter. The parameters following the class name, *cinv*, initialize instance variables within the definition of *cinv* (see below).

The logical clock, *clk*, is generated by the statement following the *conditions* declarator. The interpretation of this statement is that *clk* will oscillate between 0 and 1 with a transition occurring every 50 nanoseconds.

The procedure *cinv* shown in Figure 2 computes the input/output relationships of the inverter. The declaration

network float trigger a;

is a Network C construct that specifies that the first terminal of the inverter will be connected to a node which has a floating point value. (Network C also allows integer nodes.) The *trigger* specification indicates that the model is to be invoked every time this quantity changes state. Recall that the state is defined to be the state of the piece-wise linear model of the quantity so *cinv* will get control every time the node connected to its first terminal changes either its slope or its intercept.

The declaration

network float mos y;

indicates that the second terminal of the inverter is also connected to a floating point node. This terminal is not a *trigger* which means that the inverter will not get control if the node it is connected to changes. Also, the specification *mos* means that the node connected to this terminal is to be considered part of a MOS circuit and MOS analysis will result. This specification is required since Network C allows other types of network nodes (for example, *clk* above) which are not considered by MOS analysis. In the case of this circuit, circuit analysis will put each instance of *cinv* in a different stage.

The general rule is that the nodes appearing in the terminal list of an element in a *elements* declaration are bound in order to the *network* variables declared in the model. Similarly, the quantities in the parameter list of an element are taken in order to initialize *local* variables in the model. For example, all the instances of *cinv* in Figure 1 have the same set of parameters. They are, *vdd* (set to 5.0), *gnd* (set to 0.0), *kpu* and *kpd* (both set to 1.0e-4).

```
#define  clip(v,vl,vh)  amin1(amax1((vl),(v)),(vh))

cinv(tf,vout,iout)
float  tf;                      /* forecast time                        */
float  vout,                    /* output voltage                       */
float *iout;                    /* output current, returned             */

/*  CMOS inverter                                                       */
{
      network float trigger a;
      network float mos     y;

      local float vdd,          /* high power supply voltage            */
                  gnd,          /* low  power supply voltage            */
                  kpu,          /* k for the pull up   transistor       */
                  kpd;          /* k for the pull down transistor       */
      float       vtn,vtp,      /* threshold voltages                   */
                  vi,           /* input voltage                        */
                  ipd,ipu;      /* transistor currents                  */

/*  Threshold voltages are 0.2*Vdd.                                     */

      vtn =  0.2*vdd;     vtp = -0.2*vdd;

/*  Generate the input forecast and clip.                               */

      vi = a'*(clock + tf - a->t0) + a;
      vi = clip(vi,gnd,vdd);

/*  Compute transistor currents.                                        */

      pmos0(vi, vdd, vout, kpu, vtp, &ipu);
      nmos0(vi, gnd, vout, kpd, vtn, &ipd);

/*  Compute output current.                                            */

      *iout = ipd + ipu;
}
```

Figure 2.

The arguments to *cinv* are the forecast time and the output voltage. The output will be the output current.

The thresholds for the n and p device are computed from the power supply and the gate voltage is obtained using the linear prediction. Transistor currents are computed using the DC MOS law and summed to produce the output.

nmos0 and *pmos0* implement the continuous MOS law stated earlier for n and p transistors respectively.

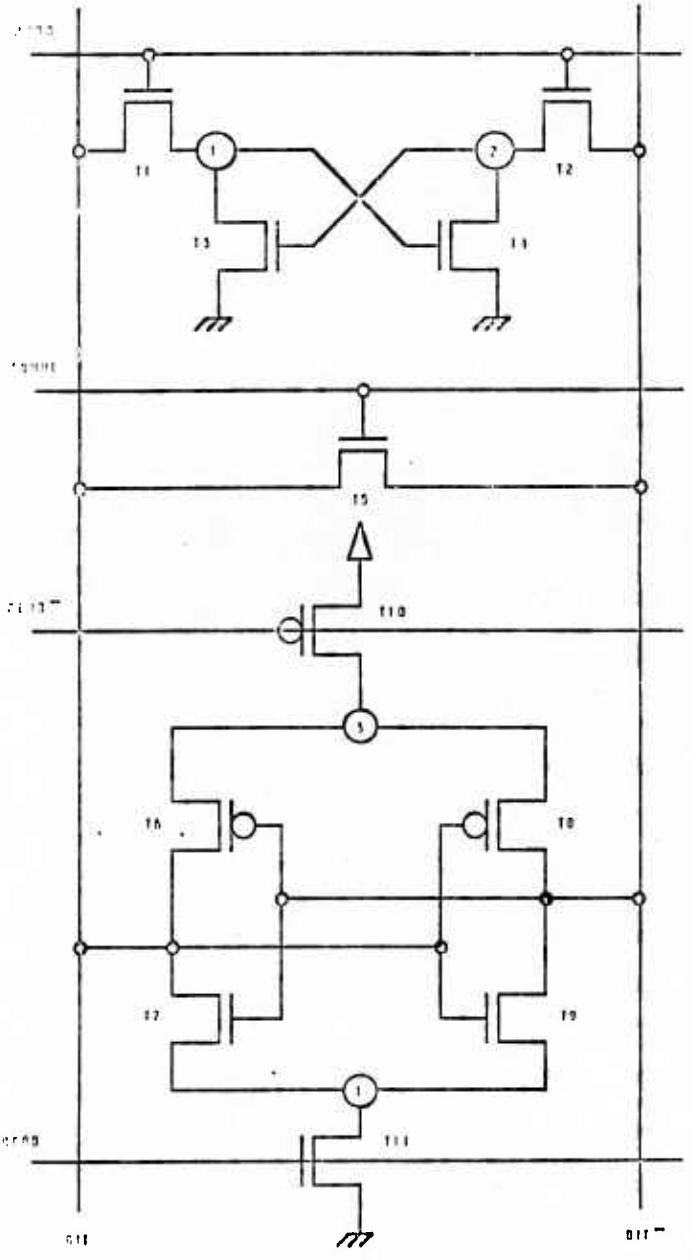


Figure 3.

The next example is the CMOS RAM cell and sense amplifier shown in Figure 3.

Since this circuit is closely coupled, circuit analysis will place all nodes and devices in the same stage. The circuit is a good example of a circuit whose behavior calculation requires the modeling of both analog and digital characteristics.

```
maincircuit ram()
{
/*  RAM cell and sense amp                                              */

      nodecap
      {
         bit = 1600.e-15;
         bitn =1600.e-15;
         n01  = 160.e-15;
         n02  = 160.e-15;
      };
      elements
      {
         g1   (gen, 0.0,5.0,1.0e9,-1.0e9)   eq,    equal, equaln
         g2   (gen, 0.0,5.0,1.0e9,-1.0e9)   clk,   read,  readn
         g3   (gen, 0.0,5.0,1.0e9,-1.0e9)   clk,   word,  wordn

         t01  (nmosfet, 2.33e-5,  1.0)      word,  bit,   n01
         t02  (nmosfet, 2.33e-5,  1.0)      word,  bitn,  n02
         t03  (nmosfet, 2.33e-5,  1.0)      n02,   n01,   gnd
         t04  (nmosfet, 2.33e-5,  1.0)      n01,   n02,   gnd

         t05  (nmosfet, 6.99e-5,  1.0)      equal, bit,   bitn

         t06  (pmosfet, 6.99e-5, -1.0)      bitn,  n03,   bit
         t07  (nmosfet, 4.66e-5,  1.0)      bitn,  bit,   n04

         t08  (pmosfet, 6.99e-5, -1.0)      bit,   n03,   bitn
         t09  (nmosfet, 4.66e-5,  1.0)      bit,   bitn,  n04

         t10  (pmosfet, 6.99e-5, -1.0)      readn, vdd,   n03
         t11  (nmosfet, 4.66e-5,  1.0)      read,  n04,   gnd
      };
}
```

Figure 4.

The circuit description given in Figure 4 is similar in form to that of the three inverters of the first example. In this case, the components consist of three waveform generators and the n and p transistors modeled by *nmosfet* and *pmosfet* respectively.

The *nodecap* declarator is used to associate specific capacitance values with nodes. (There is a small default minimum node capacitance associated with all MOS nodes which is required by the numerical method.) In this case, the *bit* and *bitn* lines are given large capacitances, 1600 femtofarads, and the memory cell storage nodes are given smaller, but larger than typical, values.

nmosfet uses the same basic DC MOS law model described above to calculate drain-source current from terminal voltages. For the *nmosfet* and *pmosfet* models, however, the gain supplied as the first parameter is one half the device transconductance, that is

$$gain = k/2.$$

The device conductance, k, is defined as the process conductance times the width over the length, that is,

$$k = k'(w/l)$$

where k' is the process conductance given by

$$k' = \mu c_{ox}.$$

μ is the carrier mobility and c_{ox} is the gate oxide capacitance per unit area.

If ϵ_{ox} is the permittivity and t_{ox} is the thickness of the gate dielectric, then

$$c_{ox} = \epsilon_{ox}/t_{ox}.$$

For this example we have taken

$\mu = 600$	$cm^2/volt\text{-}sec$
$\epsilon_{ox} = 3.9\epsilon_0 = 3.5 \times 10^{-13}$	$farads/cm$
$t_{ox} = 0.1 \times 10^{-6}$	$meters$ (1000 $angstroms$).

Therefore,

$c_{ox} = 35.0 \times 10^{-9}$	$farads/cm^2$
$k' = \mu c_{ox} = 2.0 \times 10^{-5}$	$amps/volt^2$ (approx)

and the gain parameters for the transistor models are given by

$$gain = k/2 = k'w/2l = 1.0 \times 10^{-5}(w/l).$$

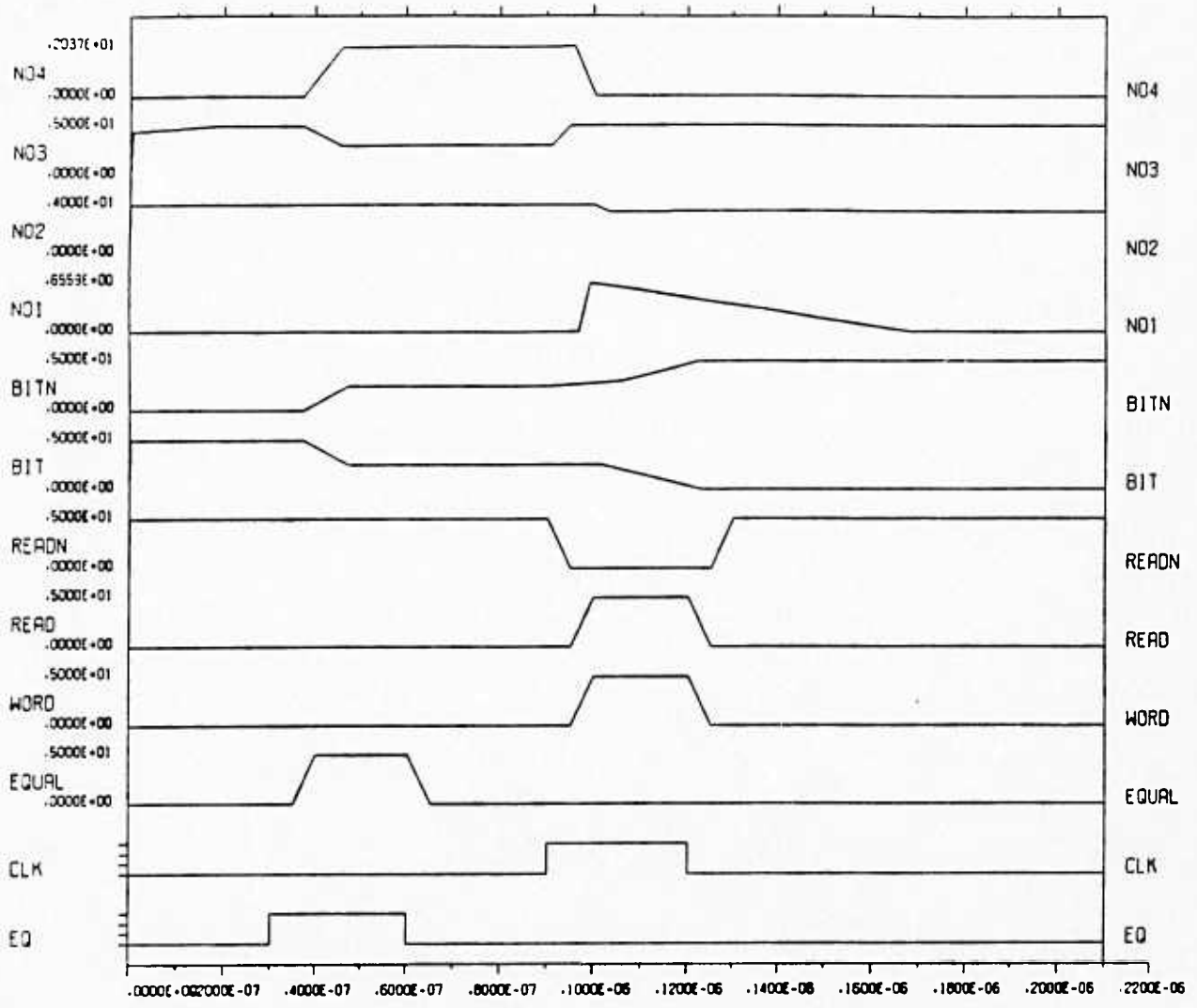


Figure 5.

The *vt* parameters are 1.0 for *n* transistors and -1.0 for *p* transistors.

Referring to the circuit diagram for the RAM cell and sense amplifier shown in figure 3, the memory cell itself consists of the four transistors, T1, T2, T3, and T4. The information stored on the gates of T3 and T4 is made available on *bit* and *bitn* when the *word* line is brought up.

Prior to bringing up the word line, *bit* and *bitn* are brought to the same potential, or equalized, by raising the *equal* line which turns on T5.

The sense amplifier, T6, T7, T8, T9, T10, and T11, consists of a pair of cross coupled inverters connected to *bit* and *bitn*. When the *word* line and *read* line are brought up

together, the amplifier accelerates the transition of the bit lines and restores the cell.

The behavior for this cell calculated by the Network C model is shown in Figure 5. In looking at this plot it is important to note that the waveforms are not all drawn using the same vertical scale. *clk* and *eq* are control signals for the waveform generators.

In this experiment, *bit* is started at 5.0 volts and *bitn* is started at 0.0 volts but *n01* is started at 0.0 volts and *n02* is started at 4.0 volts. Therefore, the state of the memory cell is the opposite of the state of the bit lines.

When the *equal* line is brought up, *bit* and *bitn* both make transitions to 2.5 volts. When the *word* and *read* lines are brought up, the state of the memory cell, represented by the charge on nodes *n01* and *n02*, is quickly transferred to the bit lines. Note that, in the process, the state of the memory cell is restored. *n02* has dropped slightly and *n01* has returned to 0.0. Note that *n01* did rise to 0.6 volts during the read when it was suddenly connected to the highly charged *bit* line.

Acknowledgements

Network C is based on work done originally as part of the author's dissertation. Because the development has been spread over a number of years, many more people have contributed than can be reasonalby mentioned here. Special thanks must be given to my chairman. Bob Herriot, and to Neil Runstein of the Academic Computer Center, both of whom have now left the University of Washington.

More recently, Rob Daasch, formerly of the VLSI Consortium, helped considerably during the development of the MOS calculation presented in this paper. Wayne Winder of the VLSI Consortium has helped in converting the system from the CYBER 855 to Unix. Also, I would like to thank Barry Jinks, VLSI Consortium liaison from Microtel Pacific Research, for his RAM cell and sense amp design which was used as the last example. Finally, I would like to thank the VLSI Consortium management, particularly Larry McMurchie, for supporting the development of this system in our current environment.

References

1. W. Beckett, *A Hybrid Paradigm for Computer Programming and Its Investigation in the Context of Electronic Circuit Simulation by Means of an Extensible Language*, Ph.D. Dissertation, Technical Report No. 78-05-02, Department of Computer Science, University of Washington (1978).

2. E. Lelarasmee and A. Sangiovanni-Vincentelli, *RELAX: A New Circuit Simulator for Large Scale MOS Integrated Circuits*, ACM IEEE 19th Design Automation Conference Proceedings (June 1982).

3. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Massachusetts (1980).

4. L. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, ERL Memo No. ERL-M520, University of California, Berkely (1975).

5. C. J. Terman, *Simulation Tools for Digital LSI Design*, Ph.D. Dissertation, Laboratory for Computer Science, Massachusetts Institute of Technology (1983).

6. Vivid Reference Manual, Microelectronics Center of North Carolina (1983).